Patent Assignment Abstract of Title

Total Assignments: 1

Application #: 09767509 **Filing Dt:** 01/23/2001 **Patent #:** NONE **Issue Dt:**

PCT #: NONE **Publication #:** 20020097269 **Pub Dt:** 07/25/2002

Inventors: Thomas Rudolph Batcha, Kipper Kyle Fulghum, Thomas Joseph Walton, Michael Thomas Juran

Title: System and method of designing, testing, and employing graphical computer code

Assignment: 1

Reel/Frame:	Received:	Recorded:	Mailed:	Pages:
011489/0795	02/08/2001	01/23/2001	04/18/2001	6

Conveyance: ASSIGNMENT OF ASSIGNORS INTEREST (SEE DOCUMENT FOR DETAILS).

Assignors: BATCHA, THOMAS RUDOLPH **Exec Dt:** 01/15/2001

FULGHUM, KIPPER KYLE **Exec Dt:** 01/15/2001

WALTON, THOMAS JOSEPH **Exec Dt:** 01/15/2001

JURAN, MICHAEL THOMAS President **Exec Dt:** 01/15/2001

Assignee: ALTIA, INC.
5030 CORPORATE PLAZA DRIVE, #200
COLORADO SPRINGS, COLORADO 80919

Correspondent: LAW OFFICE OF DALE B. HALLING, LLC
DALE B. HALLING
24 S. WEBER STREET, SUITE 311
COLORADO SPRINGS, CO 80903

Search Results as of: 4/18/2004 4:05:53 P.M.

Your search matched **21** of **1024576** documents.
A maximum of **500** results are displayed, **15** to a page, sorted by **Relevance Descending** order.

Refine This Search:
You may refine your search by editing the current search expression or enteri new one in the text box.

iconic programming Search

☐ Check to search within this result set

Results Key:
JNL = Journal or Magazine **CNF** = Conference **STD** = Standard

---

1 **Improving readability of iconic programs with multiple view object representation**
Koike, Y.; Maeda, Y.; Koseki, Y.;
Visual Languages, Proceedings., 11th IEEE International Symposium on , 5-9 1995
Pages:37 - 44

[Abstract] [PDF Full-Text (856 KB)] **IEEE CNF**

---

2 **Visual tools for generating iconic programming environments**
McIntyre, D.W.; Glinert, E.P.;
Visual Languages, 1992. Proceedings., 1992 IEEE Workshop on , 15-18 Sept.
Pages:162 - 168

[Abstract] [PDF Full-Text (452 KB)] **IEEE CNF**

---

3 **An iconic programming system, HI-VISUAL**
Hirakawa, M.; Tanaka, M.; Ichikawa, T.;
Software Engineering, IEEE Transactions on , Volume: 16 , Issue: 10 , Oct. 19
Pages:1178 - 1184

[Abstract] [PDF Full-Text (968 KB)] **IEEE JNL**

---

4 **HI-VISUAL as a user-customizable visual programming environmen**
Miller, E.; Kado, M.; Hirakawa, M.; Ichikawa, T.;
Visual Languages, Proceedings., 11th IEEE International Symposium on , 5-9 1995
Pages:107 - 113

[Abstract] [PDF Full-Text (472 KB)] **IEEE CNF**

5 **Iconic programming for teaching the first year programming sequen**
Calloni, B.A.; Bagert, D.J.;
Frontiers in Education Conference, 1995. Proceedings., 1995 , Volume: 1 , 1-
Nov. 1995
Pages:2a5.10 - 2a5.13 vol.1

[Abstract] [PDF Full-Text (356 KB)] IEEE CNF

---

6 **HI-VISUAL for hierarchical development of large programs**
Kado, M.; Hirakawa, M.; Ichikawa, T.;
Visual Languages, 1992. Proceedings., 1992 IEEE Workshop on , 15-18 Sept.
Pages:48 - 54

[Abstract] [PDF Full-Text (548 KB)] IEEE CNF

---

7 **Interpretation of icon overlapping in iconic programming**
Hirakawa, M.; Nishimura, Y.; Kado, M.; Ichikawa, T.;
Visual Languages, 1991., Proceedings. 1991 IEEE Workshop on , 8-11 Oct. 19
Pages:254 - 259

[Abstract] [PDF Full-Text (448 KB)] IEEE CNF

---

8 **Iconic programming: where to go?**
Ichikawa, T.; Hirakawa, M.;
Software, IEEE , Volume: 7 , Issue: 6 , Nov. 1990
Pages:63 - 68

[Abstract] [PDF Full-Text (556 KB)] IEEE JNL

---

9 **Designing mixed textual and iconic programming languages for nov users**
Rader, C.; Cherry, G.; Brand, C.; Repenning, A.; Lewis, C.;
Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on , 1-4 Sept. 1
Pages:187 - 194

[Abstract] [PDF Full-Text (200 KB)] IEEE CNF

---

10 **Visual dynamic environment for distributed systems**
Di Gesu, V.; Isgro, F.; Lenzitti, B.; Tegolo, D.;
Computer Architectures for Machine Perception, 1995. Proceedings. CAMP '95
20 Sept. 1995
Pages:359 - 366

[Abstract] [PDF Full-Text (836 KB)] IEEE CNF

---

11 **Dynamic interface for machine vision systems**
Tegolo, D.; Lenzitti, B.; Isgro, F.; Di Gesu, V.;
Pattern Recognition, 1994. Vol. 3 - Conference C: Signal Processing, Proceedi
the 12th IAPR International Conference on , October 9-13, 1994
Pages:323 - 326 vol.3

[Abstract] [PDF Full-Text (400 KB)] IEEE CNF

12 **Advances in visual programming**
Hirakawa, M.; Ichikawa, T.;
Systems Integration, 1992. ICSI '92., Proceedings of the Second Internationa Conference on , 15-18 June 1992
Pages:538 - 543

[Abstract] [PDF Full-Text (484 KB)] **IEEE CNF**

13 **A generic model for constructing visual programming systems**
Hirakawa, M.; Yoshimi, M.; Tanaka, M.; Ichikawa, T.;
Visual Languages, 1989., IEEE Workshop on , 4-6 Oct. 1989
Pages:124 - 129

[Abstract] [PDF Full-Text (472 KB)] **IEEE CNF**

14 **Graphical and iconic programming languages for distributed proces control: an object oriented approach**
Coote, S.; Gallagher, J.; Mariani, J.; Rodden, T.; Scott, A.; Shepherd, D.;
Visual Languages, 1988., IEEE Workshop on , 10-12 Oct. 1988
Pages:183 - 190

[Abstract] [PDF Full-Text (588 KB)] **IEEE CNF**

15 **IGIP: a framework towards open-ended visual programming**
Van Reeth, F.; Flerackers, E.; D'Hondt, T.;
Visual Languages, 1988., IEEE Workshop on , 10-12 Oct. 1988
Pages:239 - 247

[Abstract] [PDF Full-Text (672 KB)] **IEEE CNF**

1 2 Next

Iconic programming languages

One approach to the 4GL ideal that has made its way into a number of commercial products is the idea of icon based flow-chart programming. Using an authoring system of this type the developer draws a flow-chart that represents the desired sequence of actions. Authoring systems of this type are essentially very high level interpreted symbolic programming languages the symbols of which are icons that represent system functions.

The best known CBL development system of this type is the *Authorware* family of programs produced by Macromedia (ie *Authorware Professional* for *Windows* and the Macintosh, and *Authorware Star* for *Windows*). An example of an *Authorware* "Hello World!" program is shown in Figure 6.2. This program is functionally much more sophisticated than all of the other example programs shown as it contains user interaction, sound and animation. It is however by far the simplest to construct, and comparison with the complexity of the functionally much simpler program of Table 6.6 illustrates the ideal of a 4GL well.

Other authoring systems of this type are *Icon Author* (*Windows* and UNIX versions, but not Macintosh) and *Layout* (*DOS*, although it provides its own GUI).

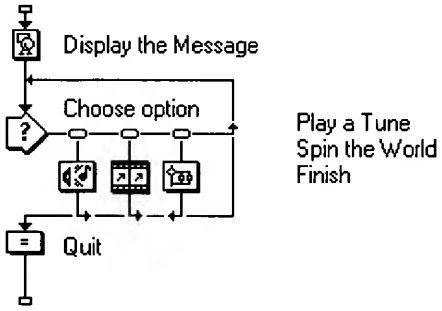

Figure 6.2. The Authorware source code for a sophisticated Windows or Macintosh "Hello World" program. When the program is run a box containing a message is displayed along with three buttons labelled "Play a Tune", "Spin the World" and "Finish". A sound is played if the first button is pressed, an animation is displayed if the second button is pressed, and the program terminates when the Finish button is pressed.

In the case of *Authorware* the programmer selects icons from a palette and places them on a bar to construct a branching flow-chart on the screen that controls the computers actions. Each icon must then be customised to suit the task in hand (eg the display icon must be told what to display, the wait icon how long to wait etc.). This process is usually accomplished either through a system of dialogue boxes or by means of a simple drawing tool. The embryonic programs can be run at any stage in their development to test their performance. When a running program encounters an empty icon, then the author will be prompted to fill in the gap. This is a powerful tool that greatly simplifies the process of courseware production, as it lends itself to the development of empty 'shells' which may subsequently be used by less skilled authors.

Although no prior knowledge of programming is required for the effective use of a system such as *Authorware* a 'programming mentality' is essential to get effective results. The icon-driven interface of *Authorware* is much less intimidating to the novice programmer than is a text based language. It is however conceptually very similar to most 3GL's as the programs consist of nested sets of routines, branches and loops, albeit that they are represented graphically on the screen.

It is also important to note that with *Authorware* although it is frequently claimed to be object oriented, it is not object oriented in the sense of OOP methodology. This means that programs tend to have a conventional rather than an object oriented structure, which is a limitation of this system (although it is a limitation common to virtually all authoring systems, iconic languages such as *Authorware* are frequently described as object-oriented and this can be misleading). A more serious limitation is that it is linear rather than event driven, and that can impose major constraints on software design.

Serious users of a system such as *Authorware* will require considerable time and effort to acquaint themselves with many of the more complex aspects of the package. Although they may well start off as non-programmers they will in the course of learning *Authorware* acquire many of the abstract skills of programming. Once these packages are mastered however it is extremely easy to construct sophisticated GUI front-ends, and although this can be extremely useful it can also be dangerous because it makes it very easy to develop bad courseware.

With this type of authoring system it is much easier to make radical changes to an application than it is with most other development tools, and it can be tempting to do this without paying proper attention to the design and pedagogical requirements of the CBL materials being produced. It is extremely important to bear this pitfall in mind when prototyping interfaces, especially when the authoring system makes changing even fundamentals very easy.

An as yet little known programming environment with huge potential for cross-platform CBL development (as well as virtually all other areas of application development) is a truly object-oriented iconic programming language called *Prograph CPX* from Prograph International in Halifax Nova Scotia (distributed in the UK by Euro Source). *Prograph* consists of an integrated environment containing an interpreter / debugger, an extensive class library that allows graphical construction of GUI front ends, and a compiler. It is currently only available for the Macintosh, although a *Windows NT* cross-compiler is promised. This will produce code which should be executable on a conventional *Windows 3.1* system using the Win32S library (this is a partial implementation of the NT API that Microsoft distributes to allow most NT applications to run under *Windows 3.1*). Prograph also intends to produce a complete *Windows* development suite, but when is anybody's guess.

The *Prograph* language is entirely iconic (there is no need for scripting at any level), using visual abstractions of data flow to represent program instructions. It is one of the purest implementations of object-oriented theory to exist in a commercial product, and the extensive class library makes interface construction easy. *Prograph* is an extremely flexible language, and it provides a serious alternative to languages such as C++ for the rapid development of sophisticated software. Substantial amounts of effort and expertise will however still be required when developing software of any complexity. It is no doubt easier to use and quicker to develop material than is a 3GL such as C, but it is much more complex than less flexible systems such as *Visual Basic* or *Authorware* (although it should be remembered that *Authorware* is very much less flexible than *Prograph*, and unlike *Prograph*, neither *Authorware* nor *Visual Basic* are true compilers).

Iconic Programming for Teaching the First Year Programming Sequence

Ben A. Calloni and Donald J. Bagert Department of Computer Science
Texas Tech University, Lubbock, TX 79409-3104
calloni@cs.coe.ttu.edu bagert@cs.coe.ttu.edu

Abstract:

Research has been undertaken to evaluate the effectiveness of using iconic (as opposed to visual) programming environments in teaching the first two computer programming courses. The co-authors have developed a Windows-based iconic programming language named BACCII. BACCII allows the user to program with icons representing all the major programming constructs and data structures within a syntax-directed environment. The user can then generate syntactically correct code for any one of several text-based languages.

This system has previously been used as a supplement to teach a Pascal-based introductory computer science (CS1) course required of both computer science and electrical engineering majors; the empirical results showed a significant increase in learning and comprehension, despite the fact that there were virtually no supplemental teaching materials for BACCII available at that time. Additional research has supplied object-oriented extensions to BACCII for use in the data structures/object-oriented programm ing (CS2) course. Current research concerns the development of a complete set of course materials for the use of BACCII in teaching both CS1 and CS2 using C++. Lecture notes, text, and lab manuals are under development. An experiment will be run using the new teaching materials at Texas Tech during the 1995-96 school year. Future research hopes to extend this program to series of pilot programs at other institutions.

Introduction

Research has been undertaken to answer the following question: *Can icon-based programming languages be used to teach first year programming concepts to undergraduate students more effectively than text-based languages?*

Many noted researchers (e.g. Glinert [4] and Scanlan [6]) have empirically established the cognitive advantage which graphical methodologies provide over textual ones. Research undertaken by the co-authors resulted in development a Windows-based iconic programming language named BACCII. BACCII (**B**en **A**. **C**alloni **C**oding **I**conic **I**nterface) allows the user to program with icons representing all the major programming constructs, such as loops, conditional branching, within a syntax-directed environment [1,2]. The user can then generate syntactically correct code for any one of five text-based languages.

This system has previously been used as a supplement to teach a Pascal-based introductory computer

science (CS1) course required of both computer science and electrical engineering majors; despite having limited teaching materials for BACCII at that time, the empirical results showed a 4-8%increase in learning and comprehension [3]. Recently, work on adding object-oriented extensions to BACCII for use in the data structures/object-oriented programming (CS2) course was undertaken.

Current research concerns the development of a complete set of course materials for the use of BACCII in teaching both CS1 and CS2 using C++. Lecture notes, text, and lab manuals are under development, and will be tested during the 1995-96 school year. This research has widespread potential significance: if it can be demonstrated that iconic languages perform significantly better than text-based languages in the learning of programming skills for a wide variety of schools and student groups, it could re volutionize how software development skills are taught in every computer-related program in higher education.

Background

The current use of C++, Ada, or any other programming language whose syntax is text-based has the major drawback that regardless of the language chosen, beginners quickly lose sight of the problem solving aspects of using an algorithm once the ``details" of the syntax begin to surface. It seems that the question should not be which language to use for a first programming course, but rather the question should be whether students can be educated in a learning environment which keeps the student focused on the problem solving aspects of the algorithm development, frees that student from the details of the syntax of any language, incorporates some software engineering concepts, and provides a gentle nudge in the area of theoretical computer topics.

It is the investigators' opinion that the most effective methodology for learning programming skills lies within the bounds of graphical representations of algorithms. Much work has been accomplished in *iconic programming* but almost all of it has been developed for high level workstations. Two factors contribute to the selectivity: speed and memory requirements. The rapid progress in high speed, high resolution, large RAM, low cost computers in the last three or four years has made possible more visually based programming environments to a wider range of users.

Such a programming environment can use a syntax-directed approach, which would guarantee syntactically correct code before the compiler even sees it. Also, by providing a graphical representation for the flow of control, one could increase semantic accuracy by accurately conveying to the user which variables of the proper type could be used at a particular point in the algorithm. The BACCII programming environment was developed with these factors in mind.

BACCII provides a standard access to code regardless of the subroutine. Click in the body section of the bit map display and the system moves to a new screen (Figure 1). In order to insure syntactic correctness, BACCII allows the student to input new statements only if they would be correct for the program's parse tree. The student selects a statement from the ``paint menu" and clicks on the <STMT> node in the display. Any selection other than a statement non-terminal is not accepted. BACCII automatica lly calculates screen locations for each new selection and scrolls the screen display down.

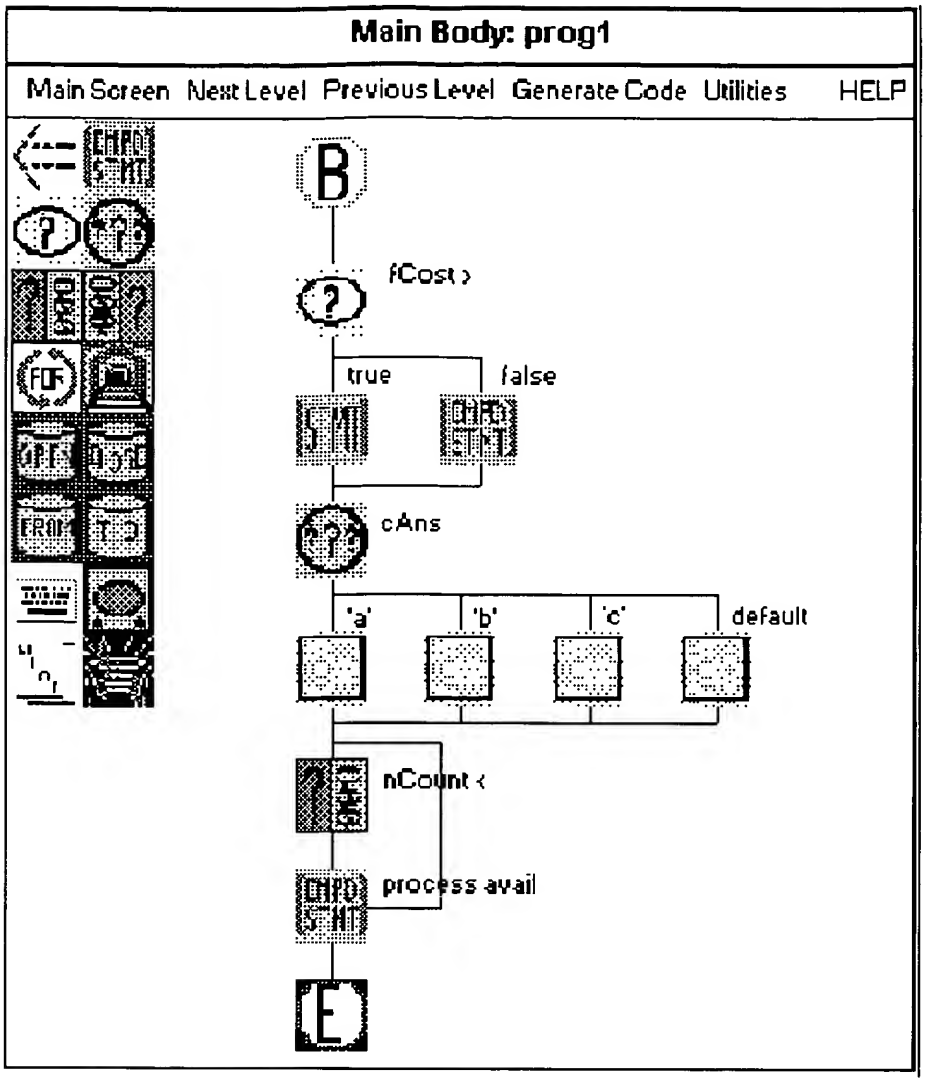

Figure 1: Coding Screen

Single statements, such as read, write, assignment, and so forth, simply replace the <STMT> icon. The selection and iteration statements always are created in groupings. For instance, one cannot create just the ``if'' part of a selection. The student *always* gets both <STMT> options for true-false. The same process is applied to all ``multiple statement'' icons.

Previous Study

In the spring of 1993, BACCII was used in the introductory programming ``(CS1)'' course at Texas Tech, Computer Science 1462 (Fundamentals of Computer Science I). Both computer science and electrical engineering students are required to take this course. The experiment was designed to divide the students into two groups: one which would use only Pascal and the other which used both Pascal and BACCII for development. The BACCII students were required to use BACCII for main programming assignments and submit BACCII files for evaluation, in addition to submitting correct Pascal code for grading. This step was necessary to insure that all Pascal students would know Pascal syntax upon completion of the course. (BACCII is robust enough that it could be used for program development without the need to learn Pascal syntax.) Use of BACCII for weekly laboratory

assignments was optional. There were almost no supplemental teaching materials for BACCII, although a tutorial was completed about two-thirds of the way into the semester.

There were several areas which were evaluated. H 0 , the null hypothesis, is used to indicate that no difference exists between the population means. H 1 is used to indicate that the BACCII group has a higher mean than the Pascal-only group.

1. H 0: The BACCII environment will result in no difference in CS programming assignments. H 0 : $\mu_1 = \mu_2$
2. H 1: The use of BACCII will result in higher scores on programming assignments. H 1 : $\mu_1 > \mu_2$
3. H 1: The use of BACCII will result in EE students having higher programming grades than CS majors. H 1 : $\mu_1 > \mu_2$
4. H 1: The use of BACCII will result in EE students having higher programming scores than non-BACCII engineers. H 1 : $\mu_1 > \mu_2$
5. H 1: The use of BACCII will result in higher scores on Lab Assignments. H 1 : $\mu_1 > \mu_2$
6. H 0: The use of BACCII will result in equivalent scores on (Pascal-only) exam scores. H 0 : $\mu_1 = \mu_2$
7. H 0: The use of BACCII will result in equivalent course scores. H 0 : $\mu_1 = \mu_2$

The analysis results are in Table 1. To summarize: for all students, BACCII resulted in higher scores on the programming assignments, labs, exams and overall course grade; the EE majors using BACCII did the same as the CS majors using BACCII; however, th ere was no significant difference in how EE students performed using BACCII vs. just Pascal.

For more information concerning this experiment, please refer to [3].

Table 1: Hypotheses Overview

Hypothesis #	H_0 or H_1	Calculated F_a or t_a F or t	Result	Confiden
1: BACCII vs. Pascal	H_0: $\mu_{bac} = \mu_{pas}$	2.68	1.960	reject $\mu_{bac} >$
2: BACCII vs. Pascal	H_1: μ_{bac} ¿ μ_{pas}	2.68	2.576	accept
3: EE BAC vs. CS BAC	H_1: μ_{EEbac} ¿ μ_{CSbac}	0.21	2.020	reject $\mu_{bac} =$
4: EE BAC vs. EE Pas.	H_1: μ_{EEbac} ¿ μ_{EEpas}	1.83	2.003	reject $\mu_{bac} =$
5: Labs	H_1: μ_{bac} ¿ μ_{pas}	2.22	1.960	accept
6: Exams	H_0: $\mu_{bac} = \mu_{pas}$	2.76	2.576	reject μ_{bac} ¿
7: Course	H_0: $\mu_{bac} = \mu_{pas}$	3.06	2.807	reject μ_{bac} ¿

Enhancements for a One-Year Computer

Programming Curriculum

Section 3 discussed Computer Science (C S) 1462, the first computer programming course for both majors and non-majors at Texas Tech. This course has its roots in the updated *ACM Curriculum 78* CS1 course, but is actually closer to the course C D 101 described on pages 103-4 of *Computing Curricula 1991* [7]. C S 1462 is a four-semester hour course with a ``partially closed" laboratory which currently uses BACCII, and has recently switched to C++ as the programming language taught.

Computer Science 2463 (Fundamentals of Computer Science II), is the second programming course; its prerequisite is C S 1462. The evolution of C S 2463 has paralleled that of 1462; it originally resembled the updated *ACM Curriculum 78* CS2 data structures course, but is actually closer to the course C D 102 described on pages 103-4 of *Computing Curricula 1991* [7]. The course is taught using object-oriented programming from the very beginning. C S 2463 is a four-semester hour course with a completely open laboratory; it has also recently switched to C++ as the programming language taught.

This course has not used BACCII, since it did not contain the necessary object-oriented extensions. However the ``visual" nature of the object-oriented paradigm (since most people view the world in terms of objects interacting with each other) makes it id eal for implementation using an iconic language. Also, it is important that BACCII be a tool that could be used beyond the introductory course. Therefore, work has been ongoing to extend BACCII so that it can be used in the object-oriented/data structures course. Since these two courses supply the student with most of the programming techniques required of the computer scientist, it is extremely beneficial that the new version of BACCII can be used for both semesters of the one-year sequence.

Course Materials for the First Year Sequence

As stated in Section 3, the initial experiment was performed with almost no supplemental teaching materials for BACCII. Recently, work has begun to develop course materials for the entire first year sequence, under a National Science Foundation Division of Undergraduate Education grant. The C S 1462 and 2463 materials will be implemented starting in the Fall 1995 and Spring 1996 semesters, respectively. Each course will have closed laboratories throughout the semester, with each lab section having no mo re than 25 students. The courses would be evaluated in a manner similar to one previously used (described in Section 3); i.e. half of the students would use BACCII, while the other half would be a control group using only C++.

The course materials that would be developed and disseminated as a result of this grant include a supplementary textbook for using BACCII with C++; a set of closed laboratories and other tutorial material for both courses, using both standard and multimed ia delivery systems; and sample on-line and off-line examination problems. Many of the course materials would be developed using the SIMPLE development environment created by Marion Hagler and William M. Marcy of Texas Tech University [5]. SIMPLE has been used successfully as an authoring tool in developing course materials for several different disciplines.

Summary and Future Directions

The BACCII iconic programming environment can be used to build programs using icons instead of the traditional text-based statements. Programs written in BACCII can be translated into several different

programming languages, including C++. BACCII has been used successfully in the introductory course at Texas Tech. Subsequent research was undertaken so that BACCII could also be used with the object-oriented paradigm. Finally, a comprehensive set of teaching materials for both courses are being developed under an NSF grant. An experiment will be run using these course materials during the 1995-96 schools.

Future work is being proposed to run experiments using the BACCII materials at five pilot schools, and to conduct additional workshops on using BACCII in the first year sequence. The authors also believe that BACCII can be beneficial in learning programmi ng skills at the primary and secondary levels; in fact, it may be even more beneficial to those students.

References

1. Calloni, Ben A. *An Iconic, Syntax Directed Windows Environment for Teaching Procedural Programming.* Master's Thesis, Department of Computer Science, Texas Tech University, Lubbock TX, May 1992.

2. Calloni, Ben A., and Bagert, Donald. BACCII: An iconic syntax-directed system for teaching procedural programming, *Proceedings of the 31st ACM Southeast Conference* , Birmingham AL, April 15-16, 1993, pp. 177-183.

3. Calloni, Ben A. and Bagert, Donald. Iconic programming in BACCII vs. textual programming: which is a better learning environment? *Proceedings of the 25th SIGCSE Technical Symposium on Computer Science Education* , Phoenix AZ, 10-11 March 1994, pp. 188-192.

4. Glinert E. and Tanimoto S. Pict: An interactive graphical programming environment. *IEEE Computer* , *17* , 11 (November 1984), pp. 7-25.

5. Hagler, Marion and Marcy, William M. Authentic and group learning technology using computer networks and intelligent tutors. *Proceedings of the 23rd Annual IEEE/ASEE Frontiers in Education Conference* , Washington DC, 6-9 November 1993, pp. 467-470.

6. Scanlan, David. Structured flowcharts outperform pseudocode: an experimental comparison, *IEEE Software* , Vol 6, No 5, Sept. 1989, pp. 28-36.

7. Tucker, Allen B; Barnes, Bruce H et. al. *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force* . Jointly published by ACM Press, New York NY and IEEE Computer Society Press, Los Alamitos CA, 17 December 1990.

Previous Up Next

mort@etp.com
Tue Oct 3 16:18:45 PDT 1995

Refine Search

Search Results -

Terms	Documents
L3 AND L2	6

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L4

Refine Search

Recall Text | Clear | Interrupt

Search History

DATE: Sunday, April 18, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=NO; OP=OR			
L4	l3 AND l2	6	L4
L3	L1 AND (translator OR translate)	36	L3
L2	L1 and object AND compile	26	L2
L1	345/763.ccls. OR 345/967.ccls.	259	L1

END OF SEARCH HISTORY